Chapter 2
Author: Rick Lukowski and Others
Compiler and Editor: Thomas W. de Boer

Freedom Post 2: Definition of Requirements

Post 2 Overview

In Post 1, we were introduced to Mills' box-structured models, and saw that Freedom adopted them as its underlying conceptual foundation for modeling software throughout the development process. In Post 2, we focus on the black box (requirements) model, and take one small step beyond Mills. The result is a giant leap for software engineering that overturns worlds of existing requirements practice, and opens a whole new world of possibilities.

Fasten your seat belt. Here we go...

Requirements Encapsulation Overview

[I thought I'd take a shot at a one-paragraph summary of Freedom. Your feedback is needed to tell me where it is too obscure, or if the whole idea is a lost cause!]

Encapsulation of requirements in code objects necessitates a change in our perspective on requirements. Rather than viewing requirements as statements _about_ the software, they are considered to be the external interface _of_ the software. The external interface consists of many stimulus-response pairs organized into into cohesive sets, each set ideally containing 7+-2 stimuli. The stimulus sets are themselves organized into a functionality tree that defines the external interface architecture. During design, the external interface architecture becomes the upper level of the design architecture. Within this upper level, a requirements encapsulating functionality module is created for each stimulus set of the functionality tree. Each functionality module contains code that: a. creates the stimulus detection mechanisms, b. listens for incoming stimuli, and c. generates the responses for one stimulus set. Changes to any requirement (external interface stimulus-response pair) are thus localized to one functionality module, making requirements change easier.

The change in requirements perspective enables requirements encapsulation because interfaces are encapsulatable whereas arbitrary statements are not. While requirements encapsulation is beneficial for reasons detailed below, is the prerequisite change in requirements perspective warranted? Is it justified by any theory of computer science or software engineering?

Definition of Requirements

We have seen that Freedom uses Mills' box-structured concepts for software engineering. In particular, we have seen that the black box model of software is used for requirements purposes. Freedom goes one step beyond Mills in the application of the black box model by defining requirements as

'The black box view of the software system.'

Standard black box theory states that the only information known about a system viewed as a black box is its external interface. Thus, the combination of Freedom's definition of requirements and standard black box theory results in a derived definition of requirements as 'the external interface of the software system.' This is exactly the definition that enables requirements encapsulation, since interfaces are encapsulatable.

Implications of the Definition of Requirements

There are many implications to this revised view of requirements.

First, and foremost, is that requirements specification becomes exactly identical to specification of the software system external interface, and ONLY the software system external interface. The definition excludes all other information.

Freedom specifies the external interface in terms of stimuli and associated responses, as proposed by Mills in http://csdl.computer.org/comp/mags/co/1988/06/r6023abs.htm. Thus defined, requirements are _always_ encapsulatable. (Note: for purposes of brevity, there is one exception that I am skipping here as it is not critical to an understanding of requirements encapsulation.)

In practice, the new interface-centric definition of requirements is consistent with the old standard definition of requirements -- 'what the system shall do' -- because all capabilities of the system MUST be reflected in its external interface. If some were not, those capabilities could not be invoked and/or could not make their results available to the users. Thus, conceptually, nothing is lost, but much is gained, by the new definition.

The downside of the new definition is that many other disparate approaches to requirements must all be sacrificed. Use cases, being usage processes and not the external interface, no longer qualify as requirements, by definition. Likewise, long lists of prose statements about the system are also no longer eligible to be called requirements. (Also, according to Parnas, 'PROSE IS THE SIGN OF AN ERROR' (his caps) in a requirements specification!) Using test cases as requirements is similarly invalid since tests are not the external interface either. Proponents of these and other previous approaches to requirements are thus faced with a difficult choice: reject their favored requirements solution, or reject the benefits of requirements encapsulation.

Benefits of Requirements Encapsulation

The primary benefits of requirements encapsulation are economic, and occur at three times: during development, during maintenance, and during reuse.

During development, the new definition of requirements needed to drive requirements encapsulation saves time and cost by replacing current requirements specification efforts with specification of the external interface. Since the external interface must be developed in any case, the current requirements effort that it replaces is time and effort saved. Also, I have

found through personal experience that specification of the external interface using Freedom's techniques of functionality trees and behavior tables to be much faster and more responsive to change than alternate ways of recording 'requirements.' Also, all information in the functionality tree and behavior tables serves directly as code specification information for the requirements encapsulating functionality modules. No additional design work need be performed for the functionality modules, resulting in further time and cost savings during development.

Even greater savings accrue during maintenance. Information exists that 60-80% of all changes during maintenance are requirements changes. Since maintenance is widely acknowledged to comprise 60-80% of all software cost, simple math indicates that 36% (0.6 x 0.6) to 64% (0.8 x 0.8, or approximately one third to two thirds, of ALL software cost is due solely to requirements changes! Thus, any technique that reduces the cost of performing requirements changes, even by a modest amount, can have a dramatic effect on the total life cycle cost of software.

The greatest potential savings of all is during reuse. As demonstrated by traditional OO technology, a fringe benefit of encapsulation is improved reusability. Likewise, requirements encapsulation should permit the creation of reusable requirements encapsulating functionality modules. In the long run, libraries of functionality modules for common external interface functionality can be developed. Such functionality might include things such as login/logout, personal data capture, inventory data capture, etc. In each of these cases, not only the functionality tree and behavior tables would be reusable, but also the implementing functionality module code along with all required support code. For example, reusable login/logout requirements would contain the necessary UI components, persistent data stores for user names and passwords, and all required password encryption. Thus, reusable requirements are fully functional subsystems. With a sufficient number of reusable requirements components, the time to construct working applications could be reduced in a manner similar to, but with much greater leverage than, is the case with current reusable Design components.

This Sounds Like ...

Web services. Indeed, web services are reusable subsystems, but differ from the Freedom concept of reusable requirements components in the following ways. First, web services apply only to stimuli originating in Web software systems, i.e., Web programmatic stimuli, whereas Freedom's reusable requirements components encompass stimuli from other software systems, human users, and/or the environment (Web and non-Web). Second, web services to date do not follow the Freedom definition of requirements, implying they do not encapsulate requirements. That is, web services are reusable but restrictive in breadth of applicability, and their requirements (exported services) are not particularly easy to change. As a general concept, web services are good, but with Freedom could be better -- more generic and less costly to maintain.

In fact, many of Freedom's requirements techniques can be found loitering in various corners of the industry. For example, the similes cohesion ideas underlying stimulus sets are also implicit in the HTML FORMS tag. A functionality tree is basically the tabular equivalent of

the AM User Interface Flow Diagram. The structure of a functionality module is identical to best accepted practice for writing Java GUI code that does NOT use inner classes (except functionality modules are not restricted to GUIs). Reusable requirements components are a functional superset and structural improvement of the web services concept.

Summary

In short, about the only things about Freedom's approach to requirements that are truly new is (1) the definition of requirements itself, and (2) the synergistic effect of combining other existing techniques under the umbrella of the new definition to achieve requirements encapsulation. As often happens with synergistic effects, the benefits of the whole are far greater than the sum of its parts. A whole new world of economic benefits becomes possible. To return to our NASA metaphor, requirements encapsulation holds the same promise for software engineering that low cost to orbit does for space exploration.

Rick Lutowski

From: Steven Gordon
Rick wrote:
> 2. Definition of Requirements

Is a specification of what the response should be for a specific stimulus or set of related stimuli not functionally equivalent to a test which checks that the system produces the specified response to a given stimulus (or set of such tests to check each of the related stimuli)?

Can we not derive any reasonable formalism for the specifications directly from such test cases in an automated fashion, if necessary? Would such test cases not also give us added benefits, such as being able to execute these tests any time and see which specifications are currently being met?

Perhaps, you will counter that Freedom derives such test cases directly from the specifications in an automated fashion and that you were going to tell us that in some future installment. If so, I find it deceptive to state up front that we have to choose between executable tests cases (the favored requirements solution for many of us) and requirements encapsulation.

Steven Gordon

From: Brad Appleton
Steven Gordon wrote:
> Perhaps, you will counter that Freedom derives such test cases directly from the specifications in an automated fashion and that you were going to tell us that in some future installment. If so, I find it deceptive to state up front that we have to choose between executable tests cases (the favored > requirements solution for many of us) and requirements encapsulation.

Rick Lutowski wrote:
> Using test cases as requirements is similarly invalid since tests are not the external interface either. Proponents of these and other previous approaches to requirements are thus faced with a difficult choice: reject their favored requirements solution, or reject the benefits of requirements encapsulation.

So, if we have a requirements 'object' per requirement, are we saying that having that object also be the test case(s) will not work? or simply that this alone is not enough, and that the rules/criteria for what is and is not in the scope or boundaries of that object are critical (perhaps the most critical?)

With a method like XP, 'requirements' are deemed to be 'specified' by the coded test-cases. The test-cases are produced during a 'task' that is part of a 'story'. So called 'stories' may get broken down into smaller items simply keep them small/simple. There is no mention of specific criteria for how to break them down, tho I imagine there must be some rule of thumb that gets used.

So with FREEDOM, who is it that gets to decide which things are part of 'this' requirement versus 'that' requirement? What defines the scope/boundary of a requirement, and how is that scope/boundary maintained so that there is the 'desired' mix of cohesion, coherence, completeness, and correctness (and minimal coupling)?

Is it possible that test-code could do this if it followed similar 'rules' for encapsulation?

In my experience using somewhat formal use-cases and a usecase-base across multiple projects, along with tracking dependencies between use-cases, I find we did some naturally occurring refactoring/restructuring of use-cases to minimize dependencies and redundancy among and across use-cases. if they'd had the 'ending structure' from the beginning (or if test-cases are refactored along with the use-cases, which then raises the issue of 'testing the test-cases' to make sure they didn't break), then I'm thinking the test-code could conceivably also serve as the requirements and still have the nice boundaries and properties you seek - but I'd have to know more about what those criteria are.

Brad Appleton

From: Rick Lutowski
Steven Gordon wrote:
>Is a specification of what the response should be for a specific stimulus or set of related stimuli not functionally equivalent to a test which checks that the system produces the specified response to a given stimulus (or set of such tests to check each of the related stimuli)?

The definition of requirements as the 'black box view of the software system' encompasses more than just responses. The full set of info includes:

1. All stimuli, including their organization, i.e., the external interface architecture

2. Externally-visible responses (but NOT internal responses, since these are not requirements by virtue of being 'inside' the black box)

3. Protocol details of the external interface, since protocol is also externally detectable info that is properly part of the system itself. Protocol includes ‘look and feel’ of the human user part of the external interface as well as comm protocol details to/from external systems and the environment.

While it is certainly possible to write tests for #2 (external responses), it is doubtful if automated tests in the traditional sense can adequately address the organizational aspects of #1 or the human factors aspects of #3. Human user/stakeholder inspection is required, which is one reason why Freedom is insistent about getting an external interface mockup into the hands of the users ASAP. It also means that, for purposes of requirements specification, Tests Are Not Enough.

Can we not derive any reasonable formalism for the specifications directly from such test cases in an automated fashion, if necessary? Would such test cases not also give us added benefits, such as being able to execute these tests any time and see which specifications are currently being met?

Let's restrict your question to those aspects of the requirements where traditional automated tests are most applicable -- #2 external response behavior.

It is my understanding that formal methods for recording external responses have been developed (in fact, quite a number of them) and many of these can support auto-generation of code from a spec captured using the formal method.

Can the same be done with code (i.e., which is really a formal method for implementation) that _tests_ the required external behavior? I hesitate to say 'no' because I have seen some pretty unbelievable things being done with generic notation translators. I will say that the problem would likely be considerably more difficult than it is for a formal method that is conceived specifically for modeling external behavior directly. With a test, the external behavior is, at best, modeled indirectly; the test algorithm (regression or whatever) is what is being modeled directly. How hard is it to infer the external behavior spec solely from the test spec (since the test spec is now ALL we have) for purposes of code generation? Probably not very easy. Maybe impossible, although that would remain for some one to prove or disprove. I do think the practical challenges are severe.

Much easier just to specify to external behavior directly. Such a spec can then be used to generate BOTH test code and the interface declarations of the app code it tests directly. If the spec changes -- regen the code, test and app. This approach produces not only runnable tests, but at least parts of the code to be tested as well. It is also more direct and less risky, i.e, simpler.

Isn't simplicity an agile principle?

Perhaps, you will counter that Freedom derives such test cases directly from the specifications in an automated fashion and that you were going to tell us that in some future installment. If so, I find it deceptive to state up front that we have to choose between executable tests cases (the favored requirements solution for many of us) and requirements encapsulation

Nothing about requirements encapsulation precludes full testing.

However, I think substituting tests for requirements precludes requirements encapsulation. If a test is the requirement, how does one encapsulate the requirement for ease of change? Remember, the test does not _implement_ the requirement, the test (requirement) only _tests_ it. Some other code needs to implement it. Keeping in mind all the code gen points above, how do we write the app code to be easy to change when the test (requirement) changes?

I don't think it can be done. If it can be done, it is certainly much more complex than Freedom's current approach, which is to use specs for specs, and tests for tests. (As Scotty says, 'Use the right tool for the right job!')

As you speculate, Freedom in fact DOES suggest creating test code from the specs. Actually, not just (1) test code, but (2) application code declarations, and (3) code documentation, similar to Java's javadoc. All this, and probably more, can be generated from a Freedom variant of a design spec that Parnas calls an 'abstract interface specification.' To avoid confusion, the Freedom variant is called a 'class interface spec.' Moreover, the toolset to do this is not unduly hard to write. I implemented a very effective Fortran version of such generators in about 6 man-weeks. I expect a version for Java would take somewhat longer to write since Java is more complex than Fortran. But conceptually, it is no more difficult.

However, your speculation that I was going to mention this later is untrue. I was NOT going to get into any of this as none of it is relevant to the primary topic of requirements encapsulation. Freedom's Implementation and Test process is two causality steps removed from requirements. I only mention it here because you asked the above questions.

* * *

It is well to keep in mind that Freedom proposes a paradigm shift in requirements. A paradigm shift is a fundamental change in some way of thinking. Thus, when thinking about Freedom's approach to requirements, and things related to it and that follow from it -- PURGE YOUR MIND!!! Do not try to relate it to XP, or TDD, or FDD, or RUP, or whatever. Read it for what it is without looking backward. Scott Ambler suggested I not compare Freedom with XP, and I think this is one reason why. You should not either.

Purge your mind. Freedom is a NEW approach. The downside is a lot of your past experience is void. The upside is a lot of your past limitations are also void. Freedom opens a new world of possibilities. I have been 'thinking Freedom' since helping to invent it almost 15 years ago, and I'm still making new and interesting discoveries about it -- things that previous methodologies never prepared me for.

We can spend the next year batting emails like the above (Freedom versus TDD), because Freedom presents an intellectual challenge to most ANY methodology you currently know. Paradigm shifts tend to do that. It is not the most productive use of our time. Please take Scott's advice and try not to compare Freedom with XP, or TDD, or whatever. Instead, try to understand it just for what is is. That will be hard enough (recall the challenge?)

Once we reach a certain comfort level, we can then compare Freedom with more fundamental concepts such as agile principles and values to see how the community might best benefit from the new possibilities it permits. This means temporarily pushing your favorite methodology to the side of your mind. For the moment, just try to understand Freedom for what it is -- a whole NEW approach with interesting new possibilities.

In keeping with my own advice to you, I will no longer provide in depth answers to questions that drag in alternate approaches. Please restrict questions to understanding Freedom, not to challenging Freedom with XYZ. That should get us to where we are going a lot faster, and it follows what Scott suggested.

Thanks.

Rick Lutowski

From: Steven Gordon
Rick wrote:
> The definition of requirements as the 'black box view of the software system' encompasses more than just responses.

I would find it easier to consider Freedom on its own terms if my introduction to it did not contain passages like the following that does compare it to other approaches:

Using test cases as requirements is similarly invalid since tests are not the external interface either. Proponents of these and other previous approaches to requirements are thus faced with a difficult choice:=A0 reject their favored requirements solution, or reject the benefits of requirements encapsulation.

Please, stick to definitions, motivations and implications, and some concrete examples. I would prefer precise definitions to simplified definitions and rambling claims (which I can only hope will be followed by complete, precise definitions).

In any case, if you do not compare Freedom to other approaches, then I will not feel obliged to counter those comparisons based on the incomplete information we have been given so far.

Steven

From: "Jason Gorman"
Rick wrote:

> The definition of requirements as the "black box view of the software system" encompasses more than just responses.

I'm a little confused as to how one could "encapsulate" a "requirement" in the implementation. Surely, if the definition of "requirement" is "the black box view of the software system" then it is externally visible and not encapsulated? You could surely only encapsulate a black box specification for one system externally in another system that communicates with it - for example, as JUnit tests. If the requirements really are encapsulated, why would you need to generate tests from them?

Also, whether or not an executable test effectively communicates the specification is down to how the test assertions are written, surely? In TDD, Kent Beck suggests that test assertions should clearly explain the intent of the method(s) under test (eg, assert(balance = oldBalance - amount) instead of assertEquals(50, balance))

Jason Gorman

From: Rick Lutowski
Brad Appleton wrote:
> So, if we have a requirements 'object' per requirement, are we saying that having that object also be the test case(s) will not work? or simply that this alone is not enough, and that the rules/criteria for what is and is not in the scope or boundaries of that object are critical (perhaps the most critical?)

I think I answered your above question in a reply to Steven, but to reiterate --

Yes,

requirement == test case

will not work in Freedom (at least, not nearly as simply as the approach it uses now.)

Also, relative to your first sentence:

There is not one requirements object per requirement. Rather, there is one requirements object per stimulus set, which may contain upwards of 10 or more requirements (stimulus-response pairs), but ideally 7+-2 (an old human factors rule).

Also, all the requirements in a requirements object are highly cohesive, since the definition of a stimulus set is a COHESIVE collection of stimuli. Thus, all the requirements encapsulated in a single requirements object are highly likely to change together. The primary goal of encapsulation -- restriction of the ripple effect of change -- is unaffected. Changing any one requirements (stimulus-response pair) directly affects only one functionality module, or requirements object.

It would be very possible to alter Freedom's encapsulation rule to result in a 1:1 mapping of requirements to requirements objects. This is actually inadvisable since (1) It ignores the natural cohesion of the stimulus sets (2) It increases the number of requirements objects by roughly an order of magnitude, resulting in what I term 'overfragmentation' of the design.

A design with too many modules is harder to maintain, just like a design with too little modularity. There is a balance point at which a certain level of modularity produces the most maintainable design. Freedom's current encapsulation rule -- create one functionality module for each STIMULUS SET -- results in designs that are pretty near the balance point consistent with good maintainability.

With a method like XP, 'requirements' are deemed to be 'specified' by the coded test-cases. The test-cases are produced during a 'task' that is part of a 'story'. So called 'stories' may get broken down into smaller items simply keep them small/simple. There is no mention of specific criteria for how to break them down, tho I imagine there must be some rule of thumb that gets used. So with FREEDOM, who is it that gets to decide which things are part of 'this' requirement versus 'that' requirement? What defines the scope/boundary of a requirement, and how is that scope/boundary maintained so that there is the 'desired' mix of cohesion, coherence, completeness, and correctness (and minimal coupling)?

Per my own reply guidelines posted in the last response to Steven, I will refrain from comments about XP or Freedom versus XP. However, you are merely using XP here as an analogy to ask a question, the question being 'how does Freedom decompose its requirements?' This is a fair question. The answer should become clear in Post 3 when we get into stimulus sets. If Post 3 does not answer things to your satisfaction, please re-raise the question(s) then. Meanwhile, please stay tuned. We'll get there.

> Is it possible that test-code could do this if it followed similar 'rules' for encapsulation?

I will not speculate on that any further than I already have, except to reiterate that Freedom dos not consider tests to be requirements. Moot point. Freedom is _not_ TDD.

> In my experience using somewhat formal use-cases and a usecase-base across multiple projects, along with tracking dependencies between use-cases, I find we did some naturally occurring refactoring/restructuring of use-cases to minimize dependencies and redundancy among and across use-cases. If they'd had the 'ending structure' from the beginning (or if test-cases are refactored along with the use-cases, which then raises the issue of 'testing the test-cases' to make sure they didn't break), then I'm thinking the test-code could conceivably also serve as the requirements and still have the nice boundaries and properties you seek - but I'd have to know more about what those criteria are.

I see what you are saying, and thus also see you are still clutching to the old requirements paradigm. Keep the above question to the side someplace and reread it somewhere around Post 6. If you've followed things up to that point, you will re-read the above and (I'll bet) grin. Most of it is red herring in Freedom -- a problem that is not a problem.

Again, Freedom is a completely new way of thinking about requirements. Do not try to map Freedom to past experience with Use Cases, tests-as-specs, or anything else except one, because it doesn't map. That one exception is interface-first. If you have ever built a system starting with the external interface and working 'inward', that would likely be a valid experience in helping understand Freedom. Purge all other approaches from your mind. They don't map.

Thanks for the good questions. Again, stay with us and all of them should be answered in due course.

Rick Lutowski

From: Brad Appleton
Rick Lutowski wrote:
> There is not one requirements object per requirement. Rather, there is one requirements object per stimulus set, which may contain upwards of 10 or more requirements (stimulus-response pairs), but ideally 7+-2 (an old human factors rule). Also, all the requirements in a requirements object are highly cohesive, since the definition of a stimulus set is a COHESIVE collection of stimuli. Thus, all the requirements encapsulated in a single requirements object are highly likely to change together. The primary goal of encapsulation -- restriction of the ripple effect of change -- is unaffected. Changing any one requirements (stimulus-response pair) directly affects only one functionality module, or requirements object.

Okay - so what I'm getting at is, how do you KNOW that? What are the criteria used that will tell me that a particular stimulus should go in set A rather than set B. What are the different kinds of relationships between stimuli and which ones are desirable/undesirable w.r.t. promoting better encapsulation, abstraction, and information hiding

(note that those are three different things IMHO, albeit strongly related: Abstraction decides which things of an entity in the problem domain are contextually relevant to warrant being in the solution domain, encapsulation decides how to package those things together in the solution domain so that the 'right' have the 'right' visibility/access, and information hiding is one particular encapsulation technique)

What are the different kinds of coupling and cohesion w.r.t. stimuli and stimuli-sets (Meiler Page Jones would simply use terms like connascence and encumbrance and their various subtypes)? How does one identify and recognize them? What are the criteria for knowing if a stimuli-set will have better encapsulation or worse encapsulation if a include a add/remove a given stimulus from the set?

It would be very possible to alter Freedom's encapsulation rule to result in a 1:1 mapping of requirements to requirements objects. This is actually inadvisable since (1) It ignores the natural cohesion of the stimulus sets (2) It increases the number of requirements objects by roughly an order of magnitude, resulting in what I term 'overfragmentation' of the design. A design with too many modules is harder to maintain, just like a design with too little

modularity. There is a balance point at which a certain level of modularity produces the most maintainable design. Freedom's current encapsulation rule -- create one functionality module for each STIMULUS SET -- results in designs that are pretty near the balance point consistent with good maintainability.

Okay - I guess I misread the page on your website that talks about 'Freedom from Traceability' due to a 1:1 mapping between requirements objects and requirements. Glad to see some higher-level granularity is being used, but then I assume the traceability is to requirements objects (rather than individual requirements) and while a particular requirements object can be traced to a particular stimuli-set, trying to trace a particular element of code or design to a possible subset of a stimuli-set is deemed unproductive and unnecessary because the stimuli-set is 'closed' due to its organization according to the criteria that tell us it is well encapsulated, loosely coupled, and highly cohesive.

So with FREEDOM, who is it that gets to decide which things are part of 'this' requirement versus 'that' requirement? What defines the scope/boundary of a requirement, and how is that scope/boundary maintained so that there is the 'desired' mix of cohesion, coherence, completeness, and correctness (and minimal coupling)? Per my own reply guidelines posted in the last response to Steven, I will refrain from comments about XP or Freedom versus XP. However, you are merely using XP here as an analogy to ask a question, the question being 'how does Freedom decompose its requirements?'

Yes. I'm after the elements of organization (the stimuli) and their boundaries, the types of relationships between them, and the criteria used to group them together into sets that are 'closed' under the various 'requirements operators', and then how one knows what the boundaries of these sets are, and when their encapsulation criteria/conditions are weakened, strengthened, sealed, or broken.

> I see what you are saying, and thus also see you are still clutching to the old requirements paradigm.

Gosh that sounds incredibly condescending and superior. I don't think you have enough information in the above to make any such judgement. Doing so comes across as belittling dismissal that tries to invalidate anothers thoughts, and is ill-advised.

I am trying to use analogy as a basis for understanding here. use-cases are analogous to your requirements objects at the very least to the extent that they are a mechanism for grouping together sets of otherwise 'atomic' individual requirements, and trying to inter-relate them. There may certainly be aspects of them that go beyond that which don't map to freedom. But do not attempt to throw out the baby with the bathwater by trying to cast aside valid bases for analogy and comparison and do not presume to tell us it is completely invalid. Rather, find the level/ways in which there is basis for comparison and the identify the ways where there is not.

> Again, Freedom is a completely new way of thinking about requirements. Do not try to map Freedom to past experience with Use Cases, tests-as-specs, or anything else except one, because it doesn't map.

Um, trying to tell someone to purge everything they know/think is likely not going to be effective. The human mind cannot help but to attempt to understand the unfamiliar in terms of what is currently familiar, and then 'chunking' and understanding the differences between them.

I don't suspect that I or others are looking to find an exact map. We are looking for a point of comparison to understand not just similarities to but also differences from things we currently are familiar with. Do not try to make it an all or nothing comparison.

So rather than telling folks to forget/ignore what they know or think and suggest it is inferior/invalid, I would strongly urge instead using that as a resource to leverage increasing understanding rather than invalidating it. Talk first about the ways in which it is similar, and then they ways in which it is different. Use them as analogies into others experiences to reach shared meaning and then extend it by first sharing where it does hold and then showing where it does not. I think you will find that far more successful in increasing our understanding rather than telling us we are 'clutching to the old'

> Purge all other approaches from your mind. They don't map.

NO! It is *your* burden to find a way to map them, and to show where and how and when it fits and does not fit. Please do not tell me what to do or how to think. I wish to be enlightened, I have no need to be mentally deprogrammed and reprogrammed. That is not a requirement for successful understanding, and the more you insist upon it the less success you will have here.

I realize you want to try and start from ground zero and make a clean slate and build upon that. But there is no way you are going to stop people from relying upon other things they have already experienced. Telling them to purge/forget/ignore those things and let you reprogram/rewire their thinking will not be effective and will not win people over. You are going to have to work with those things rather than push them aside in order increase understanding, and use those as stepping stones for understanding differences and similarities. Otherwise you simply increase the likelihood of disconnect and empty-intersection and communication and understanding will be the victim.

If you want to have success in communicating Freedom you will have to find a way to work WITH the thoughts and ideas we already have, and acceptingly step into our worlds/views to bridge the gap from them to your own. If you want us to embrace and understand and demonstrate respect and appreciation for your thoughts and views, you will have to do the same with ours and use those as a starting point rather than as a discard pile.

Brad Appleton

From: Rick Lutowski
Brad Appleton wrote:
> Okay - so what I'm getting at is, how do you KNOW that? What are the criteria used that will tell me that a particular stimulus should go in set A rather than set B. What are the

different kinds of relationships between stimuli and which ones are desirable/undesirable w.r.t. promoting better encapsulation, abstraction, and information hiding

These sorts of questions will be answered starting in Post 3.

> Okay - I guess I misread the page on your website that talks about 'Freedom from Traceability' due to a 1:1 mapping between requirements objects and requirements. Glad to see some higher-level

As you now know, 1:1 is not completely accurate. The true relationship is N:1 (N requirements : 1 object). On the web site, N:1 would have simply raised more questions than it answered, and 1:1 communicated the concept better. (Also, it COULD be made 1:1 if someone really wants, so the site isn't lying either.)

granularity is being used, but then I assume the traceability is to requirements objects (rather than individual requirements) and while a particular requirements object can be traced to a particular stimuli-set, trying to trace a particular element of code or design to a possible subset of a stimuli-set is deemed unproductive and unnecessary because the stimuli-set is 'closed' due to its organization according to the criteria that tell us it is well encapsulated, loosely coupled, and highly cohesive.

Yes, there is _implicit_ traceability from the req spec to the implementing functionality modules. This traceability is, in fact, 1:1 at the stimulus set (SS) level, just not at the individual stimulus-response pair level, where it is N:1.

Explicit traceability is never tracked in Freedom. Not necessary due to the straightforward 1:1 SS mapping.

Yes. I'm after the elements of organization (the stimuli) and their boundaries, the types of relationships between them, and the criteria used to group them together into sets that are 'closed' under the various 'requirements operators', and then how one knows what the boundaries of these sets are, and when their encapsulation criteria/conditions are weakened, strengthened, sealed, or broken.

Yes, that is all Post 3 (stimulus sets) and 4 (functionality trees) So far we have just covered the basic concepts and definitions. The real operational meat, which is what your questions are getting at, start in the next post.

> > I see what you are saying, and thus also see you are still clutching to the old requirements paradigm.
> Gosh that sounds incredibly condescending and superior. I don't think you have enough information in the above to make any such judgement. Doing so comes across as belittling dismissal that tries to invalidate anothers thoughts, and is ill-advised.

Not condescending, just stating it the way I read it.

I will say for the record right now that I will be the last to disparage anyone on this list. If anyone reads any of my statements that way, you are reading it wrong. Look for an alternate interpretation.

Your statement about encouraging readers to start with a 'clean slate' is exactly what I intended by 'purge your mind', not disparagement of what you now know.

> I am trying to use analogy as a basis for understanding here. use-cases are analogous to your requirements objects at the very least to the extent that they are a mechanism for grouping together sets of otherwise 'atomic' individual requirements, and trying to inter-relate them. There may certainly be aspects

Yes, that analogy with use cases is a reasonable one. It was not originally clear to me that that is what you were driving at.

Note here we are using 'use cases' in the looser 'chunk of user functionality' sense of the term, not in the stricter 'UML notation' definition of the term.

Um, trying to tell someone to purge everything they know/think is likely not going to be effective. The human mind cannot help but to attempt to understand the unfamiliar in terms of what is currently familiar, and then 'chunking' and understanding the differences between them.

I learned a little about the psychology of paradigm shifts from attending various TQM type courses. One of the things that struck me was the statement that those who have the hardest time adapting to the new paradigm are the current experts in the old paradigm. They go into a number of examples to illustrate the point, such as the Swiss watch experts failing to grasp the significance of digital watch technology, resulting in Japan wresting the market from the Swiss; things like that. Well, at least one reason the 'most expert' are at risk is they have the hardest time purging their mind (or starting with a clean slate). Since their mind is so full of hard-earned knowledge, their vested interest is also higher, so they have the most to loose from the new paradigm. So the experts either ignore it, or resist it, because they are unwilling or unable to purge/wipe clean/start over.

Here, no one is ignoring or resisting, but we are all learning. Granted, one way we learn is by mapping things to our existing knowledge base, as you point out. But how does one map digital technology to a knowledge base founded on springs and gears? How does one map interface-centric requirements to requirements techniques based on prose statements? Sometimes the mappings just aren't there, or are so weak as to be ineffective as a learning mechanism. In such cases, is is often more fruitful to abandon mappings and analogies as a learning mode and just try to grasp the ideas directly.

When we get to Post 3 and actually start to answer many of the questions you raised above re stim set cohesion and the like, I think you will find that the direct understanding approach will, in fact, be the easiest.

Nothing in Freedom is complex. Keep in mind that it is a minimalist methodology. Like any other methodology, it can be bloated into a fat pig of gov't proportions in the name technical or academic elegance. I know because in the time period from 1991-1992, soon after we invented it, I started enhancing it with all manner of elegant features. But when I started using Freedom myself on my own projects, well, you should have seen all the crud fly overboard! By the time it was down to where I would use it, it was dirt simple. All the basic infrastructure -- concepts, definitions, stim-response, etc -- was still the same. But all the make-work was purged out. It was lean and mean, easy to understand (once the mind was cleared) and, most importantly, so easy to use that I (yes, lazy hacker me) would actually use it because NOT using it was more work.

The point of this little historical snapshot is that Freedom is not complex. Mappings and analogies to things like Use Cases and TDD are not necessary to understand it, and are more likely to hurt than help comprehension. I do not think that grasping the concepts directly will pose much of a problem for anyone on this list.

Sure, Freedom was originally developed for NASA, but in its current form it is NOT rocket science. It's fundamentally a concept model and some definitions that support recording techniques for specifying external interfaces, because it views the external interface as requirements.

We will start covering those recording techniques, and answering your many astute questions, in Post 3.

Rick Lutowski

From: Ashley McNeile
Rick wrote:
> Encapsulation of requirements in code objects necessitates a change in our perspective on requirements. Rather than viewing requirements as statements _about_ the software, they are considered to be the external interface _of_ the software.

There is an interesting discussion on this in Michael Jackson's book 'Problem Frames' (ACN Press 2001) in Chapter 1, section 1.7: 'The problem is not at the interface'. His point is that the external interface of the software is sometimes not the right focus when specifying requirements, as a proper understanding of requirements can depend in subtle ways on deeper aspects of the problem domain.

Ashley

From: Rick Lutowski
Ashley McNeile wrote:
> There is an interesting discussion on this in Michael Jackson's book 'Problem Frames' (ACN Press 2001) in Chapter 1, section 1.7: 'The problem is not at the interface'. His point is that the external interface of the software is sometimes not the right focus when specifying

requirements, as a proper understanding of requirements can depend in subtle ways on deeper aspects of the problem domain.

I've previously received vague reports that there might be some anti-interface line of thought out there, but you are the first to provide a solid link to it. Will try to track down Jackson's book and see what he says. (Hey, how much can an over-the-hill rock star know about software engineering, anyway?!-)

Thanks very much for this interesting tidbit.

Rick Lutowski

From: Hubert Matthews
Ashley McNeile wrote:
> There is an interesting discussion on this in Michael Jackson's book 'Problem Frames' (ACN Press 2001) in Chapter 1, section 1.7: 'The problem is not at the interface'. His point is that the external interface of the software is sometimes not the right focus when specifying requirements, as a proper understanding of requirements can depend in subtle ways on deeper aspects of the problem domain.

Jackson says that requirements relate to the business problem, which involves the complete system and not just the machines. The interfaces with the machines are specifications. As an example, he mentions a patient-monitoring system. The problem isn't about the interface between the machine and the patient (i.e. the sensors), it's about the patients and their vital signs.

Hubert Matthews

From: Rick Lutowski
Hubert Matthews wrote:
> Jackson says that requirements relate to the business problem, which involves the complete system and not just the machines. The interfaces with the machines are specifications. As an example, he mentions a patient-monitoring system. The problem isn't about the interface between the machine and the patient (i.e. the sensors), it's about the patients and their vital signs.

It's ironic he should use that example, because my wife is a registered nurse. I hear about what the problem is all the time, and it's _not_ the patient's vital signs. From the nurse's perspective, the problem is companies that build medical equipment with frustrating user interfaces that are cumbersome and time-consuming to use. The nurses know darn well how to do vital signs, give medications, hang drips, etc. and just wish the computers would let them do it and get out of their way. As it is, patient care is suffering because too much time is being spent wrestling with ill- thought out computer interfaces.

Prior to responding to this post, I double checked with her. She verified that in her 30+ year nursing career, which spans hospitals from DC to California and Michigan to Texas, neither she nor any nurse she has known has _ever_ been consulted by a medical equipment manufacturer about the design of a new piece of equipment. In the same 30+ years, she recalls nurses being consulted by an architectural firm about the design of a new unit maybe once or twice. In the medical area, based on her experience at least, the engineering community, esp the software engineering community, is sorely out of touch with its users.

Jackson doesn't know what he's talking about if he thinks vital signs are the problem. The problem is, indeed, lousy interfaces, borne of lack of communication with the end users. I've been hearing about it for years.

As a software type, esp one who has helped invent a way to develop software based on paying up-front attention to the user interface, this has been a real thorn in my side. I feel there should be something I can do to help her and her fellow nurses. However, whenever I look for jobs at medical equipment firms, it just has never clicked. Oh, well.

Thank's for that interesting synopsis, Richard. I think I'll save my money on Jackson's book.

Rick Lutowski

From: Ashley McNeile
Rick Lutowski wrote:
> I hear about what the problem is all the time, and it's _not_ the patient's vital signs. From the nurse's perspective, the problem is companies that build medical equipment with frustrating user interfaces that are cumbersome and time-consuming to use.

With respect, I think you are missing the point. A system with a lousy user interafce can still meet its business requirements. It's just difficult to use.

> Jackson doesn't know what he's talking about if he thinks vital signs are the problem.

I think that is unlikely. I would recommend that you take the trouble to read what he has written on this subject.

Ashley

From: Richard Fisher
Rick Lutowski wrote:
> 2. Definition of Requirements

Given Freedom's definition of requirements:
- Where/How does Freedom deal with persistence requirements?

- Where/How does Freedom deal with business rules that are not externally apparent:
    - Value derivation rules for values not displayed
    - State change rules
- Where/How does Freedom handle providing data to support analytic needs (i.e. OLAP stuff)?

I may be getting ahead but, how do you encapsulate requirements that involve maintenance of data as part of one set of stimuli and analysis/reporting of that data as part of very different stimuli?

I hope these questions are in the proper flavor of this discussion.

Rick Fisher

From: Rick Lutowski
Richard Fisher wrote:
> Given Freedom's definition of requirements:
- Where/How does Freedom deal with persistence requirements?

If the persistent store (DBMS or whatever) is external to the black box, i.e., is part of another external system black box, then persistence is treated as external system stimuli and responses. Requests to the external persistent store would be be triggered by other external stimuli, such as a 'save' button press from a human user, or the stimuli could be initiated from the external persistent store to our black box, e.g., signaling the store has some new data for us.

If the persistent store is internal to the black box, i.e., it is our responsibility to develop and is not 'given' to us by someone else who is responsible for it, then the persistent store is internal design (gray box) information and not requirements. The details of interfaces to the store are then part of the design, not part of the requirements. It is logical that they should be so, since we now have full control over the design and implementation of the persistent store.

Thus, the black box boundary that defines the requirements is really a responsibility boundary. Others, such as Roger Sessions, have also claimed it is a trust boundary. I agree with this assessment as well.

>Where/How does Freedom deal with business rules that are not externally apparent:
    - Value derivation rules for values not displayed
    - State change rules
- Where/How does Freedom handle providing data to support analytic needs (i.e. OLAP stuff)?

Anything that is not externally visible or detectable is design or implementation information, not requirements, by the box-based definitions. This does not preclude the customer from specifying 'internal' things be done a certain way. Certainly the customer can specify whatever s/he wants -- it's their money and their system. However, under Freedom, the

developers would treat such information as ‘design and implementation (D&I) constraints’ rather than requirements. The customer may not care much about the distinction, but because different encapsulation strategies are involved for requirements (external interface) and design decisions ('common service' data and algorithms), it makes a difference to the developers.

Note that state changes may be internal or external. Internal state changes, if specified by the customer, are D&I constraints. Externally-visible state changes are part of the required response behavior to the triggering stimulus.

OLAP is no different conceptually than any other kind of algorithm. Data required by it that comes from the external interface is part of the requirements. Some of the OLAP characteristics, such as fast response, can also be captured as requirements because response time is externally detectable. Other characteristics, such as shared data security, are D&I constraints except for any externally- visible components (e.g., public keys). A good way to handle something of a 'standard' nature like OLAP is to specify use of some reusable commercial package that supports it. 'Vendor X OLAP' would then be a D&I Constraint if the package were contained wholly internal to the black box, or as a requirement if external such as in the form of a web service. In either case, the plethora of detailed specs that might be required by a concept like OLAP are then simplified to a single standard term, e.g., ‘OLAP.’ This is nothing new; we do it all the time with prose requirements. References to standards can also occur in Freedom, such as in the response behavior specs.

Bottom line is that the black box boundary is definitive -- all parts of the system to be developed that comprise the boundary (interface) are requirements. Things inside the boundary are design and implementation info.

Ideally, all D&I info inside the boundary are the development team's responsibility to define any way they think best. With 100% freedom of design behind the black box boundary, the team theoretically can produce the best system for the given schedule. As soon as the customer starts specifying D&I constraint in addition to actual requirements, design choices pass out of the developers hands, and along with them the possibility of producing the technical best system. But the world is never ideal. Customers always seem to have their reasons for specifying information internal to the black box, i.e., are willing to give up a certain amount of technical excellence for other reasons, -- standardization, prior agreements with vendor B, etc. If the customer is a bit open-minded, it should be possible to tell them ‘this is a D&I Constraint, not a requirement’ to make them aware of the implicit tradeoff effect of their request. If the custoemr is merely micro-managing, this might be enough to get him to back off. (I know, good luck!)

> I may be getting ahead but, how do you encapsulate requirements that involve maintenance of data as part of one set of stimuli and analysis/reporting of that data as part of very different stimuli?

The stimulus-response pairs involved with the maintenance of the data, say a DBMS admin console, would be encapsulated as one set of functionality modules in one part of the functionality tree (external interface architecture). The S-R pairs involved with analysis and

reporting, say a sales staff's screen set, would be encapsulated as a different set of functionality modules in another part of the functionality tree. Meanwhile, the services common to and used by both, such as app specific JDBC-DBMS access modules, would be implemented, not as functionality modules, but as design decision-hiding modules just like in current OO.

It may be worthwhile to note here that Freedom does not alter current OO, but rather extends it to include requirements. In so doing, it moves the external interface to the front of the development process where it becomes one of the first objects of focus for the customer and developer alike when developing new software.

<aside> The specification of the external interface is up front in the development of NEW software, but is not the first step in Freedom. The first step in Freedom is the 'Build versus Buy process, which attempts to meet customer needs via COTS rather than new construction, if possible. Only if this fails does specification of the external interface for new software commence, if authorized by the customer.

Also, prior to initiation of the Build versus Buy process, Freedom must be initiated. Freedom assumes it is part of a larger customer or corporate process for process improvement. Decisions relating to what part of the corporate process should be automated are assumed by Freedom to be made as part of this corporate process improvement process. This might be as formal as a CMM process, or as informal as some manager getting a 'budget and a brainstorm.' However, it happens, Freedom's view is that such decisions are outside the realm of any software development process such as Freedom, but rather are within the realm of some corporate management process. Thus, specification of many types of 'requirements' such as 'business requirements' or 'enterprise requirements' are considered by Freedom to be out of scope of software development and properly within the scope of corporate IT management's process improvement process, as well. </aside

> I hope these questions are in the proper flavor of this discussion.

Yes, these were exactly the types of questions I had anticipated. Thanks for asking them.

Rick Lutowski

From: Rick Lutowski
Jason Gorman wrote:
> I'm a little confused as to how one could 'encapsulate' a 'requirement' in the implementation. Surely, if the definition of 'requirement' is 'the black box view of the software system' then it is externally visible and not encapsulated? You could surely only encapsulate a black box specification for one system externally in another system that communicates with it - for example, as JUnit tests. If the requirements really are encapsulated, why would you need to generate tests from them?

Sorry it has taken so long to reply to you question. During the years in which I have been using and refining Freedom, I asked and answered the same question in my own mind.

However, you are the first 'other' person to ask it, so this is the first time I have ever had to write the answer down. It is a non-trivial question and the answer is likewise non- trivial. So it took a while, and I still probably didn't do a very good job. However, I will send it as it is and leave it to your further questions to tell me where it is insufficiently clear.

This is a very important question, and I apologize for the length of the answer. As Churchill once said 'I didn't have time to make it shorter.'

* * *

Types of Encapsulated Information

Parnas identified three major types of information that can be encapsulated for ease of change: interfaces to hardware, design decisions, and required behavior. The exact approach varies slightly for each case. However, the goal is identical in all cases -- restricting the ripple effect of a change to a single code module. That is, when a single item of information changes -- be it an interface to hardware, a design decision, or a required behavior -- only a single implementing code module need change with it.

Approach to Hardware Encapsulation

[feel free to skip if you know how hardware encapsulation works]

In the case of interfaces to hardware, the approach is to define a generic interface representative of all hardware of that type (such as printers). The application is then built to the generic interface, rather than to the actual vendor interface of the specific hardware. Finally, a 'hardware-hiding' module, or 'device driver,' is built to convert messages to the generic interface into messages to the vendor interface of the specific hardware (HP Printer). When the hardware changes (HP is replaced by Canon), a new hardware-hiding module (device driver) for the printer (Canon) is written. The remainder of the app is totally unaffected by the change. The success of this approach depends on the robustness of the generic interface. For example, if the specifies of the generic interface used 2D point arrays in the external interface, then someone tries to replace the existing 2D device with a 3D device, the process fails. The generic interface will need to change to support the 3D device, and the entire app with it (as least those parts that call the hardware-hiding module, which we assume are well- enough scattered to present a major maintenance headache.) Thus, it is critical to make the generic interface sufficiently robust ('stable') to be able to handle all likely hardware changes.

Approach to Design Decision Encapsulation

[feel free to skip if you know how data encapsulation works]

In the case of design decisions, there are two major types: code (algorithms) and data.

In the case of code, the approach is easy. Go to the method and change the algorithm. Assuming the old and new algorithms use the same data, that's about all there is to it. The change is restricted to the one method.

In the case of data, the approach gets stickier because data is typically shared among many methods. The approach here is to collect cohesive data together into a single group. The favored criteria for data cohesiveness is the concept of an 'object,' i.e., the data together models some real world object or logical thing. All the methods that perform operations on that data, such as fetching it, or setting/computing it, are collected together with the data. The collection of data and associated methods are then walled off from the rest of the application using syntax like a class declaration (in Fortran 77, lacking class declarations, I used a separate directory plus a subroutine naming scheme to achieve a similar effect.) All other parts of the app that need the data of that object are prohibited from accessing the data directly. They must go through the methods of the object, which perform the access function for them and send the results back to the requester. When a datum changes (int changes to a float to handle fractional vacation days, or whatever) the data structure of the object is changed along with the affected methods of the object. Since the object's methods are the only ones in the entire app that actually access the datum, no other methods of the app need change. The success of this approach depends on the robustness of the interface to each method of the object. If, for example, the interfaces of some of the methods needed to change to return a float instead of an int when the datum changed from int to float, then the process breaks. All the methods of the app that call the methods of the object with changed interfaces also must change. We assume such methods are well-enough scattered to present a major maintenance headache. Thus, it is critical to make the object interface sufficiently robust ('stable') to be able to handle all likely changes to the data structures of the object.

Approach to Requirements Encapsulation

In the case of changes to requirements, the first problem is to understand the nature of requirements in sufficient detail so as to permit their encapsulation. This means an understanding of requirements more precise than 'what the software shall do,' since that definition is inadequate to guide creation of requirements encapsulating code. The approach chosen by Freedom is predicated on requirements being defined as 'the black box view of the software system,' i.e., the software system external interface. Since hardware-hiding clearly demonstrates that interfaces are encapsulatable, defining requirements in terms of the software system interface appears to offer adequate precision.

However, certain attributes of requirements (now defined as the software external interface) make it clear that the requirements encapsulation approach must be different from both hardware-hiding and data-hiding:

Volatile info	Serves	Stable element
hw interfaces	internal modules	driver interface
data structures	internal modules	module interface
requirements	external users	???

What should the stable element for requirements encapsulation? Since the requirements do not serve modules, offering a stable internal interface to the program's modules does little good. Also, creating a stable external interface in front of the true external interface (per the hardware hiding approach) cannot be the answer because that is equivalent to changing the requirements, which is not an option. We must encapsulate the requirements we are given, because we expect them to change (not due to our 'encapsulation needs', but due to the real needs of the users!)

Clearly, the stable element cannot be a stable interface, per the other forms of encapsulation.

The clue to the stable element is in the goal -- to restrict the ripple effect of a change to a single module. The goal would be met if each requirement were implemented by a single module. Current design strategies do not do this, since the implementation of a single requirement often spans a large number of modules.

However, if we could invent a design strategy and accompanying architecture in which each requirement was implemented by only one module, then that would meet the goal. When a requirement changes, only its one implementing module would need to change. Since a requirements encapsulation module does not provide services to other modules, there should be little risk of the change rippling throughout the program. (If there were, the usual data-hiding approach could be applied to a requirements encapsulation module as well.) As it turns out, the external interface-centric definition of requirements permits a design strategy in which each requirement can be implemented by one module. The key lies is in capturing certain relevant information about the requirements (i.e., the external interface.)

The process for requirements encapsulation is to define the external interface in terms of stimulus-response pairs, as proposed by Mills. The stimulus-response pairs are then grouped into cohesive sets based on their role and organization in the external interface. These stimulus sets are then organized into a tree-like structure called a functionality tree, which defines the architecture of the external interface as a whole. Requirements encapsulation is then accomplished by directly mirroring the black box architecture of the external interface in the gray box architecture of the program modules. That is, each branch in the external interface architecture has a matching branch in the module architecture. Each leaf (stimulus set) in the external interface architecture has a matching leaf (requirements encapsulating ‘functionality module’) in the module architecture.

This design strategy not only results in the desired goal of each requirement being implemented by a single module, but also establishes a near-perfect architectural symmetry between requirements and design and code. When one learns the organization of the program from the user's perspective, one is also learning the organization of the program from the developer/maintainer's perspective because the two are identical (at least, at the level of the functionality modules). This symmetry further enhances the goal of ease of change of requirements because requirements, changes by definition, will be changes to the external interface (stimuli, responses, and protocol/L&F). Since the architectures are the same at all 'abstraction' levels, knowing where in the external interface the change occurs tells the developers and maintainers exactly where in the module architecture the change must occur. Even better, with no stable interface involved, there are no stable interfaces to break due to

lack of sufficient robustness. What _can_ break requirements encapsulation is not conforming to the stable element, which is nothing more than the requirements encapsulation design rule -- 'create one functionality module for each unique stimulus set in the functionality tree.'

Volatile info	Serves	Stable element
hw interfaces	internal modules	driver interface
data structures	internal modules	module interface
requirements	external users	REQ ENCAP DESIGN RULE

Fortunately, it should be entirely feasible to create a tool set to generate code declarations for the functionality modules directly from the external interface specifications according to the design rule. With the process automated, the rule, and requirements encapsulation, should never be broken.

Rick Lutowski

From: Philippe Back
Maybe a sample case is in order... my head is spinning with all this discussion & not a single sample !

Philippe

From: Rick Lutowski
Philippe Back wrote:
> Maybe a sample case is in order... my head is spinning with all this discussion & not a single sample !

Yes, right on! Examples are needed.

First, though, recall where we are in the 'grand plan' --

Post 1: concept models
Post 2: definition of requirements <-- WE ARE HERE>
Post 3: stimulus sets
Post 4: functionality trees
Post 5: unique (requirements reuse)
Post 6: behavior tables
Post 7: functionality modules

So far, we have just covered concepts and definitions; all theoretical stuff intended to provide (1) background, (2) rationale, and (3) common nomenclature and semantics, for the remaining Posts. The actual applied topics, the 'how to', starts with Post 3. Examples will start then too, so things should start getting much more applied and much less theoretical commencing with Post 3.

One interesting thing is that, because Freedom defines requirements as the externally visible parts of the system, requirements are always reverse-engineerable, at least in principle. This is true even if the source code has been lost, if no requirements were ever formally recorded, etc. In practice, one is constrained by the parts of the system one can actually interact with. For example, the human users of a system can reverse engineer the human user interface, but may not be able to reverse engineer the interface to an external data base (at least, not without instrumenting the external data base to capture traffic to the system.)

We can use this to advantage for our example purposes by choosing a real system and analyzing and recording its requirements using Freedom's techniques. I am still thinking about what might make a good example for the group. It has to be (1) simple yet not trivial, and (2) available or accessible to the entire group. Candidates might be some simple free software program everyone can download, or it could even be a simple web site somewhere. I am thinking in terms of the latter.

If anyone has any ideas or druthers, please let me know.

Rick Lutowski

From: Brad Appleton
Rick Lutowski wrote:
> I am trying to use analogy as a basis for understanding here. Use-cases are analogous to your requirements objects at the very least to the extent that they are a mechanism for grouping together sets of otherwise ‘atomic’ individual requirements, and trying to inter-relate them. There may certainly be aspects.

Yes, that analogy with use cases is a reasonable one. It was not originally clear to me that that is what you were driving at.

> Note here we are using 'use cases' in the looser 'chunk of user functionality' sense of the term, not in the stricter 'UML notation' definition of the term.

Actually I'm using it in a much looser sense than that. I'm looking at what the basic-idea of a use-case is and what it tries to be at the conceptual level. Any specific assumptions about a particular implementation of use-case methodology should not hold here. Don't assume UML or RUP or so-and-so's book about use-cases. Just go back the basic idea in Jacobsen's original writings (and again look at the basic idea and not the formal implementation specifics).

From that point of view, a use-case attempts to capture the key elements of an interaction dialogue between the system and an actor/agent that gives input to the system for a particular purpose. Its not so specific as to be a single ‘scenario’ or ‘trace’ and in fact does cover multiple ‘families’ of scenarios/traces that result from an interaction for that purpose. For the moment, I don't care how the information in the use-case is stored or formatted (regardless of whether it is prose or some formal-spec, much less what the format of the prose or spec is). And there are some defined relationships between use-cases such as uses, extends, generalizes and 1-2 others.

From this perspective, a use-case defines a particular agent/actor (could be a person, could be something else), a particular business goal or purpose for the particular system interaction, and a 'scope' (boundary) that gives at least some basic guidance in what should be part of the use-case and what should not.

I think the important thing is that the use-case is something that is more coarse-grained than a single requirement (just as a module/class is more coarse-grained than a single method) and there should be a single coherent purpose that cohesively organizes and orders all the things it contains. Plus there is a way to relate these entities (use-cases) together at their boundaries (rather than relating the innards of things between them).

Either way, I ultimately have a requirements grouping mechanism that lets me think about and manipulate requirements solely at the 'modular' level and a criteria for why each 'requirement' is part of one group rather than another group.

I think it is a mistake to throw use-cases out the window as invalid/inapplicable. Before throwing it and other approaches 'out the window' lets focus not on the specific details and implementation of those approaches but on the basic ideas and concepts. If we focus on the details you will find reasons to discard everything under the sun and leave no room for common ground. If we focus on the essential ideas (the abstraction level) I think there is much knowledge and experience that can be kept and applied as a basic for understanding and compare/contrast.

So please let's focus on the 'idea' level rather than the a particular instantiation when dealing with comparisons to other known things in one's experience. I get the impression you like to focus on the detail-level at which things can be precise and unambiguous. And that is perhaps why you are so quick to advise 'purging ones mind'. I think one can 'free one's mind' of those detail-levels without having to purge it, and that the abstract conceptual level is a perfectly acceptable and reasonable place to start bridging the understanding-gap rather than insisting on 'clean slate'. It means having to tolerate variance and ambiguity between ideas and definitions to the extent that one can gain a basic 'foothold' and using the details to refine what the differences are and where.

I realize that kind of ambiguity and tolerance for it appears contrary to what Freedom seems to value. At the same time, in order to communicate a method such as Freedom I suspect it becomes more important to be able to tolerate and communicate those kinds of things that the method itself eschews in order to 'get the basic idea' and build up on it from there.

> How does one map interface-centric requirements to requirements techniques based on prose statements?

I think you can do it the way I just did above. Focus on the abstract level of the basic idea and show which parts of it are important or applicable (the modular grouping mechanism, the criteria for coherence/cohesion) and which ones aren't (the prose or format). That leaves you with the 'common ground' from which to fill in the blanks, but at least you have a place to start filling them in form and know where they need to be filled in (in this case, the details of

the criteria, the granularity and shape of the elements inside that are being grouped, and the form/format of what they are)

> Sometimes the mappings just aren't there, or are so weak as to be ineffective as a learning mechanism. In such cases, is is often more fruitful to abandon mappings and analogies as a learning mode and just try to grasp the ideas directly.

I think that is sometimes true. I think it usually is not. I think instead we need to first try harder to look at the level and context of the comparison and adjust that first, and the first thing to go are low-level details so we can even see if we're in the same ballpark or not. Maybe when comparing roadmaps, before trying to look at all the side-streets and driveways and rest-stops, first just look at the freeways and highways and ignore the rest. At that point, not only can it become feasible and useful to compare maps and territories, it can even work when the other map isn't even roads, but instead waters and rivers and streams. Then they are just topologies over different spaces but with some similar relationships and configurations between them that serve some basic properties and are still a valid basis of communication and understanding for BOTH similarities and differences.

Focus first on the similarities, at whatever levels possible, then clarify differences. Or as Bruce Lee says ‘steal what is useful and discard the rest’ (just because some or even all of the details are useful doesn't mean the entire ideas are useless).

>... and the like, I think you will find that the direct understanding approach will, in fact, be the easiest.

I am willing to bet that is not at all the case. And is only true because you insist it is so and insist on considering all the details when mapping/comparing and discarding unless all of them match. Need to allow that ‘fuzzy’ pattern matcher to do some work.

> Mappings and analogies to things like Use Cases and TDD are not necessary to understand it, and are more likely to hurt than help comprehension.

Whether or not it is necessary or useful is entirely up to the person attempting to do the understanding. Best to let them decide that. I strongly suspect that it does NOT hurt more than help and that has only been your experience and your way of thinking about it. And when it comes time to do comparison/contrasting as you have asked, particularly to judge whether or not it is indeed agile, some of it will be unavoidable.

So please - don't tell me how I think. Don't tell me how to think. Don't tell me what I may and may not compare/contrast against in my own mind, and don't try to tell me I can't find anything useful out of doing so. I'm quite confident I will prove you wrong every single time when it comes to what *I* think and what *I* know. Instead of wasting your time trying to tell me those things, and adding no substantive content in the process, instead spend that effort looking at the abstract idea level and saying what is similar and what is different rather than ‘don't think that way’.

Brad Appleton

From: Brad Appleton
Rick Lutowski wrote:
> Fasten your seat belt. Here we go...

Locked and ready Cap'n ...

> Encapsulation of requirements in code objects necessitates a change in our perspective on requirements. Rather than viewing requirements as statements _about_ the software, they are considered to be the external interface _of_ the software.

How can we better differentiate the 'external interface' from a 'user interface'? I know in some methodologies it is deemed important to identify the 'system boundary' (I think Derek Coleman's FUSION did this, I think even OMT did it back in the days before RUP/UML).

The system/software will exist in a certain context and environment. The ways in which the system interacts with that context and environment are (I suspect) the 'external interfaces' you mean. The context helps us decide what things are relevant (or not) to a particular purpose, and hence guides us in knowing which things to keep and which to eliminate when forming an abstraction, as well as which things to hide or not when encapsulating the abstraction as a modular entity.

Peter Wegner has some papers on 'interaction paradigm' of computing that are worth reading w.r.t. the above.

> The external interface consists of many stimulus-response pairs organized into into cohesive sets, each set ideally containing 7+-2 stimuli.

Sounds (again) a lot like interaction systems. This can be regarded as vectors or tuples where the initial stimulus is something from the context+environment, and the response comes from the system (at least initially). After that, subsequent stimuli to the response could be from either the system or its environment+context or a combination thereof (and so could the response) and those non-initial stimuli and responses could each be expressed as a vector showing the component of each that they contain: (context, environment, system) So initial (pure) stimuli might always have '0' as the third component and non-zero in at least one of the other two, and initial (pure) response might always have 0 in both of the first two and non-zero in the third component.

How then does one define a relationship between a pair or stimulus-response pairs ((sc1, se1, ss1),(rc1, re1, rs1)) and ((sc2, se2, ss2),(rc2, re2, rs2)) so that one can test the relationship for strength/weakness of coupling and cohesion?

> The stimulus sets are themselves organized into a functionality tree that defines the external interface architecture. During design, the external interface architecture becomes the upper level of the design architecture.

Sounds similar to the way that a ‘functional baseline’ is intended to map to an ‘allocated baseline’ in basic CM theory.

> Within this upper level, a requirements encapsulating functionality module is created for each stimulus set of the functionality tree. Each functionality module contains code that: a. creates the stimulus detection mechanisms, b. listens for incoming stimuli, and c. generates the responses

Sounds like the same basic idea of ‘test first’ to me to the extent that you are basically creating codified executable environment in which to simulate the system's environment and context, and generate system expected responses. At some point you need to define/invoke the system interface, and you are forcing that to be at the stimulus points (I think - yes?). Where test-first starts to violate this (be different) would seem to be when it starts to make assumptions about code-level interfaces.

It also bears a resemblance to a message-based type of thinking, in the sense of how the Schlaer-Mellor method used the term, where the message would be completely decoupled from the ‘method’ (or implementation) and in fact could be ‘thrown over the wall’ to a messaging and lookup service (I believe the used to term ‘wormhole’) that could take care of the details of binding it to an implementation at ‘translation time’).

>... for one stimulus set. Changes to any requirement (external interface stimulus-response pair) are thus localized to one functionality module, making requirements change easier.

I'm not yet convinced. I see how a requirement specification change impacts only one ‘functionality module’ and hence only one stimulus-entry point and response-exit point. That is only evidenced at the boundaries of the system (I think I prefer the term ‘system boundary’ to ‘external interface’ for my own usage). I don't see how it guarantees against ripple-effect of code implementation changes once inside the system.

You have effectively defined and characterized the system boundary, and (lets assume ideally) partitioned the boundary into disjoint sets of ‘service areas’ (bigger than a ‘service’, smaller than a ‘component’). The fact that you do this effectively as a black-box only limits localization/impact to the entry and exit points, and (if done as a true black box) merely guarantees that your requirements objects and service-area entry/exit points cannot discern whether or not ripple-effect changes are necessary further inside the system itself. So there must be some additional mechanisms and policies responsible for that.

> The primary benefits of requirements encapsulation are economic, and occur at three times: during development, during maintenance, and during reuse.

How might this promote reuse of requirements across project existing in parallel within an organization? Lets take the example of an IT shop that might have scores of smallish projects and products with lots of potential overlap/redundancy in many aspects of requirements in several projects (differencing perhaps only by certain technology assumptions/restrictions/interfaces).

And how might such reuse impact a given stimulus-response set? Would it always be reused in whole? or would it ever happen that in the present of a similar but different context and similar but different environment (or even different context but same environment) that what makes a stimulus-response set 'cohesive' for one project might be less so for the other one it is being reused in, enough so that I need to modify and/or select only part of it in order to reuse it?

Seems to be in that case one gets into issues of ontologies, similar to how web-services (e.g. ebXML) does. And when the reuse context can differ, it means you may have to do some kind of refactoring of your requirements objects and of the stimulus and responses across and between stimulus sets. How would you do that? And how do you encapsulate the requirements so as to minimize the impact of such a change (plus preserve the integrity and correctness when doing it)

>... following ways. First, web services apply only to stimuli originating in Web software systems,

Web-services do service-orientation in general does not.

>... web services to date do not follow the Freedom definition of requirements, implying they do not encapsulate requirements.

Nor are they supposed to. They are supposed to encapsulate the service (and its mechanisms/providers) the service, not its specification or definition. However the can also encapsulate the service interface to the extent that a service can be queried by name and purpose (e.g., one such implementation might use JavaBus and JavaSpaces) and the interface to invoke can be 'discovered' at run-time by the portal/wormhole mechanism that decouples the message from its sender *and* its service provider (and possible

I see that you have a way of attempting to encapsulate specifications of behavioral system interaction. That would be akin to encapsulating the specification and even interface of a service itself. It doesn't guarantee the implementation mechanism impacted by the requirements change is itself well encapsulated and that the impact is localized.

If we're getting precise about how to formally characterize a requirement (e.g. as a stimulus-response set), then should we attempt to do the same for a requirements change? Can we assume it always affects only one stimulus-response set? What about the kinds of requirements changes? There would be changes that simply altered the combination or permutation or sequencing of existing stimuli and responses.

What about changes that change the context/environment itself and not merely add/change/remove responses, but indeed would add/change/remove stimuli and even possible dependencies and interactions between environment+context that would change the entire topology of an existing decision-tree for all changes/added/removed stimuli from their respective stim-response sets? (isn't this exactly the kind of thing that autonomic computing would need to deal with most?)

>... (exported services) are not particularly easy to change. As a general concept, web services are good, but with Freedom could be better -- more generic and less costly to maintain.

As a general comment, I would recommend refraining from claims such as the above and instead say more about how/why. Otherwise it sounds like 'marketing' that tries to impress rather than inform, and express rather than explain. Informative explanations will be more enlightening and persuasive to me, whereas the latter may 'rub' many the wrong way.

>... in various corners of the industry. For example, the similes cohesion ideas underlying stimulus sets are also implicit in the HTML FORMS tag. A functionality tree is basically the tabular equivalent of the AM User Interface Flow Diagram.

So the whole thing can just be one big 'active/dynamic' XML file. Lets make a wiki out of it! :-) We can define tag for stimuli, responses, and sets of them, and style-rules for how to format them, and then to respond to some of that. Seems like FitNesse already can do some of this ;-)

Brad Appleton

From: Rick Lutowski
Brad Appleton wrote:
> How can we better differentiate the 'external interface' from a 'user interface'?

They are almost the same thing, with one difference which has to do with sources of stimuli. Stimuli can originate from any of three places:

1. humans. e.g., a GUI
2. external systems., e.g., a network
3. the environment, e.g., a sensor

Stimuli that come from 1 and 2 can be thought of as coming from users (human or programmatic). Stimuli that come from 3 are not from users. That is, the environment (air, physical forces, simple passage of time, etc) are not 'users' in the generally accepted sense of the term, but are sources of stimuli. Hence, 'external interface' is a superset of 'user interface,' with the difference being the stimuli from the environment.

Freedom assumes the black box does not send external stimuli to itself. Any messages sent and received by the system itself are considered internal, and thus are not requirements stimuli but rather messages among individual modules.

> Sounds (again) a lot like interaction systems. This can be

Could be, but Freedom is not mathematically formal about it. As I said before (I think), Freedom is simple in application, not mathematically formal, so as to be easily usable by mathematically unsophisticated developers. It's very possible interaction theory, formal

methods, or both could be applied to make Freedom more mathematically elegant. I am not convinced such would improve it in practice, although it might in theory.

> The stimulus sets are themselves organized into a functionality tree that defines the external interface architecture. During design, the external interface architecture becomes the upper level of the design architecture. [...] Sounds similar to the way that a 'functional baseline' is intended to map to an 'allocated baseline' in basic CM theory.

A functionality tree has absolutely nothing to do with CM. It is a recording notation for capturing with minimal effort the relationship of the stimulus sets to one another. Like any other notation, including code, the FT can be placed into a CM system, but that is a totally different issue.

> Each functionality module contains code that:
> a. creates the stimulus detection mechanisms,
> b. listens for incoming stimuli, and
> c. generates the responses

Sounds like the same basic idea of 'test first' to me to the extent that you are basically creating codified executable environment in which to simulate the system's environment and context, and generate system expected responses.

Not a simulation of anything. These are things the requirements encapsulating 'functionality modules' must do to implement the requirements (external interface.)

> So the whole thing can just be one big 'active/dynamic' XML file. Lets make a wiki out of it! :-) We can define tag for stimuli, responses, and sets of them, and style-rules for how to format them, and then to respond to some of that. Seems like FitNesse already can do some of this ;-)

Are you serious or being facetious? If the latter, it is totally out of character with your other comments.

Rick Lutowski

From: Brad Appleton
Rick Lutowski wrote:
> A functionality tree has absolutely nothing to do with CM.

How about with the kind of 'decision' tree Parnas describes in his papers on program Families and designing software for ease of extension and contraction?

>> So the whole thing can just be one big 'active/dynamic' XML file. Lets make a wiki out of it! :-)
> Are you serious or being facetious?

A bit of both actually (call it ‘whimsical’ :). I do think that FitNesse currently has the capability to do some of this, but you'd still have to teach it a few things, and use the formalisms you describe rather than the prose. But there is a lot that is already there.

Brad Appleton

From: Rick Lutowski
Brad Appleton wrote:
> How about with the kind of ‘decision’ tree Parnas describes in his papers on program Families and designing software for ease of extension and contraction?

Later on I think he changed the name to 'module hierarchy,' In any case, Parnas' tree or hierarchy reflects the DESIGN architecture. Freedom's functionality tree defines the REQUIREMENTS architecture. Conceptually they are different due to being at these different levels of abstraction. However, Freedom then gets clever by invoking symmetry to jumpstart the design architecture by seeding it with the requirements architecture, so the RA becomes dual purpose, if you will. But this does not change the fact requirements and design architectures are different logical entities.

Rick Lutowski

From: Brad Appleton
Rick Lutowski wrote:
> Later on I think he changed the name to 'module hierarchy,' In any case, Parnas' tree or hierarchy reflects the DESIGN architecture. Freedom's functionality tree defines the REQUIREMENTS architecture. Conceptually they are different

Of course. Again I see you are focusing on detail implementation level rather than the basic idea and seeking differences in detail rather than identifying commonality in concept. Again, look at the organizing and governing criteria and principles in the new context (requirements instead of design) and please try looking up instead of down first (it’s a lot less condescending that way too :))

The decision tree Parnas describes is one of design decisions that partitions the design space and identifies design paths based on constraints and tradeoffs made at various points. Now look at that again but focus on the idea and the structure rather than the ‘design’ word. Replace design with ‘functionality’ or ‘thing’ or ‘requirements’ if you wish - I really don't care. The important thing is not so much the things that are being organized and partitioned and grouped but HOW it is being organized and partitioned and grouped.

Remember we are talking about things like encapsulation and coupling and cohesion and abstraction and information hiding. The main difference is the context in which they are being applied. Most of the literature talks about how to apply those things to design elements in design space, and you are of course now trying to describe applying those things to requirements elements in requirements space. So taking every analogy/comparison to ways of

doing those things in the design space and saying ‘its different because its design’ is not very meaningful or useful and sees only the obvious and ignores the important.

If were gonna talk about how to do those things (e.g. encapsulation) and the criteria for how and when and why, you're gonna have to be able to compare and contrast them to the same kinds of criteria from a design context and how the translate to a requirements context, and do so by focusing on the structure and principles and how they can translate to the new context rather than saying ‘different context - does not compute’)

I guess I should apologize, because I feel I've asked that of you so many times already that I am frustrated by the fact that I keep having to ask it and keep having to remind you and you keep not doing it. I wonder if that is an inability or an unwillingness - either way it could be some valuable feedback as to why you may have had little success in trying to convey these things before in any other terms besides trying to go ‘clean slate’ and asking the listener to discard what they know.

I think you are missing a vital opportunity to leverage what they know, and to also more readily ‘connect’ with them rather than turn them off by asking them to turn off their experience and chalk it up to ‘knowledge’ about paradigm-shifts to justify doing so when in fact it really is not necessary and can be an even more effective tool in bridging that understanding gap (translating the content of what they know into the new context instead of trying to discard it because of the old context - seems to me maybe it is not the listener that is the clinging to the old so much as the instructor that is focusing on it when you could instead help them focusing on the context-translation for the existing mental schema rather than discarding it and trying to reconstitute it from scratch)

So again - I ask the same question. ‘How about with the kind of 'decision' tree Parnas describes in his papers on program Families and designing software for ease of extension and contraction?’ - and again I would note that I am asking about the tree and the kind of tree, not so much the things that the tree is organizing and dividing.

Are you in fact creating a kind of tree (in which the elements are functional beasties rather than design beasties) that partitions functionality space into their corresponding ‘modules’ for ease of requirements extension and contraction? Are you not creating on overall ‘functionality family’ at multiple levels of scale - first with stimulus SETs, then with black-boxes of stimulus-sets, and then again treating the ‘system’ itself as an overall black-box, and then (again, from a subsequent post) actually breaking up the system itself so that any ‘interactions’ between parts of the systems are instead viewed as interactions between black-boxes (which seems more component-based than systems-based, but that is another matter)

Brad Appleton

From: Rick Lutowski
Brad Appleton wrote:
> Are you in fact creating a kind of tree (in which the elements are functional beasties rather than design beasties) that partitions functionality space into their corresponding ‘modules’ for

ease of requirements extension and contraction? Are you not creating on overall ‘functionality family’ at multiple levels of scale - first with stimulus SETs, then with black-boxes of stimulus-sets, and then again treating the ‘system’ itself as an overall black-box, and then (again, from a subsequent post) actually breaking up the system itself so that any ‘interactions’ between parts of the systems are instead viewed as interactions between black-boxes (which seems more component-based than systems-based, but that is another matter)

You have asked me to back off on implementation details and compare Freedom's techniques with other techniques at a higher level. Now I must ask you to also back off, not in a technical sense, but from the perspective of the presentation schedule to the group.

In previous posts you asked for clarification about stimulus sets, which is scheduled for Post 3. Now you are asking similar questions about functionality trees, which are scheduled for Post 4.

Hey, we are still on Post 2!

Meanwhile, some in the group report their heads swimming in concepts and high level discussions with no examples to illustrate any of it.

Thus, for the sake of others in the group, I must ask that you please be patient wrt future topics, and focus on the topic at hand. Once the group has completed its questions and comments on the current topic, we will move on to Post 3 and explain stimuli and stimulus sets, which will include examples. Hopefully, this will start to clarify matters for you and many others as well.

Thanks for your patience.

Rick Lutowski

From: Pete C Ruth
I have been watching the interactions regarding Freedom, with particular regard for the communication and methodology mismatches that have been discussed.

I have faced a similar problem for quite some time regarding the ‘model’ that I created to enable application systems to be developed and deployed ‘better and faster’. When I describe it, and the development process that goes with it, I usually get responses that clearly indicate that I am not getting my message across to my audience (developers or business folks, or both). It was quite a frustrating problem, and I had experience enough in both realms to have axpected that problems of this nature would not occur. And I must admit, I did not handle it as well as I could or should have, although this was not visible to the audience. Today, I have the luxury of sitting down with a potential client and demonstrating the facilities, allowing them to ask whatever questions that come up, and cover everything in a manner that satisfies their curiosity. I do have a ‘program’ that I follow, a sort of guide, that I go over at the outset, but the client is free to ‘guide’ the demonstration of meeting until we have covered all the bases, or I remind them of the topics still to cover (or not, depending upon the situation).

Regarding Freedom, I am acutely interested in it, and I am acutely impatient to get to the ‘nuts and bolts of it’. While I am able to get to the meat of something pretty well by reading the subject matter, I learn best and most quickly when there are examples that reinforce what I'm reading. So as you can imagine, I'm itching to see some diagrams. My interest is further piqued, by the way, because of several informal conversations I had with Harlan Mills in the early 1980s regarding the development ‘model’ and process that I had developed. These conversations were, according to Dr. Mills, ‘mutually beneficial, informative and great fun’. I am embarrassed to admit that I did not fully appreciate his stature in the field as well as I might have, but that seemed to be not uncommon, owing to his personable and unassuming nature (at least in his presentations). I'm sure I still have a tablet full of notes somewhere from these presentations, as well as the ‘free for all’ discussions that followed.

For those of you who are not old enough to have been around during the tumultuous days of ferment in our field, when just about anything you did was likely being done for the first time, ever, by anyone, I wish there were some way to convey the excitement and wonder at the way things were then. Many of the emerging technologies of today had their roots in some ‘you bet your company’ (or project, or job, or whatever) project from those time, and earlier. And I have to tell you, one of the most difficult impediment to keeping up with many of these new things is the change of descriptive terms (probably another good reason to employ lots of diagrams and examples in presentations). Is it absolutely necessary to re-invent the language with every new way of doing things?

Anyway, that's my rant for this month. Guys like me would like pictures, diagrams and a continuing flow of ‘meat’. It's nothing personal. It's probably in our genes, along with a strong affinity for highly caffeinated beverages, donuts and other programming fare, and an aversion to sunlight. Go figure.

So for my money, I'm not complaining, I'm just impatient.

Regards,

Pete

From: "Jason Gorman"

Actually, I'm just drunk... Pls move on to the next installment :-)

Meanwhile, some in the group report their heads swimming in concept and high level discussions with no examples to illustrate any of it.

Jason Gorman

From: argnosis
PeteCRuth wrote:

> I have been watching the interactions regarding Freedom, with particular regard for the communication and methodology mismatches that have been discussed.

You are not alone in that. I reckon there are a few people whose interest in this site is based on a fundamental knowledge that there is something rotten in the state of IT. This is one of the few forums which allow you to challenge the current paradigm without being drowned out by conservative flame. There are very few forums where Rick would have been able to get as far as he has. Great to see. Perhaps your and my and Rick's view of IT coincide at a level which does not yet have the semantics for us to see it. Perhaps the discussion Rock is having will lead us to clear terminology. Perhaps that terminology will be good enough to swing the rest of IT through the paradigm shift(s).

Argnosis

From: Rick Lutowski
Jason Gorman wrote:
> Actually, I'm just drunk... Pls move on to the next installment :-)

PeteCRuth wrote:
> So for my money, I'm not complaining, I'm just impatient.

I expect to have Post 3 out by tomorrow evening.

Rick Lutowski

From: Philippe Back

Would you mind having a list of the posts put together somewhere on your website for easy reference ?

Thanks!

Philippe Back

From: Rick Lutowski
Philippe Back wrote:
> Would you mind having a list of the posts put together somewhere on your website for easy reference ?

See http://www.jreality.com/freedom/papers/req_encap/posts.html

Also, Thomas has offered to put the postings together into a composite which should be more complete than my 'quick and dirty' above. (I assume he will come out from behind the shadows when he is ready;-)

Rick Lutowski